МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ

федеральное государственное автономное образовательное учреждение высшего образования

«САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ   
АЭРОКОСМИЧЕСКОГО ПРИБОРОСТРОЕНИЯ»

КАФЕДРА КОМПЬЮТЕРНЫХ ТЕХНОЛОГИЙ

И ПРОГРАММНОЙ ИНЖЕНЕРИИ (КАФЕДРА №43)

КУРСОВОЙ ПРОЕКТ  
ЗАЩИЩЕН С ОЦЕНКОЙ

РУКОВОДИТЕЛЬ

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Старший преподаватель |  |  |  | Е.О. Пятлина |
| должность, уч. степень, звание |  | подпись, дата |  | инициалы, фамилия |
| ПОЯСНИТЕЛЬНАЯ ЗАПИСКА К КУРСОВОЙ РАБОТЕ | | | | |
| ПРОЕКТИРОВАНИЕ ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ ИНФОРМАЦИОННОЙ СИСТЕМЫ «АГЕНСТВО НЕДВИЖИМОСТИ» СРЕДСТВАМИ ЯЗЫКА UML | | | | |
| по дисциплине: ОБЪЕКТНО-ОРИЕНТИРОВАННОЕ ПРОЕКТИРОВАНИЕ ИНФОРМАЦИОННЫХ СИСТЕМ | | | | |

РАБОТУ ВЫПОЛНИЛ

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| СТУДЕНТКА ГР.№ | 4831 |  |  |  | К.А. Корнющенков |
|  |  |  | подпись, дата |  | инициалы, фамилия |

Санкт-Петербург 2020

Оглавление

[ВВЕДЕНИЕ 3](#_Toc40106122)

[Проектирование программного обеспечения информационной системы «Агентство недвижимости» 4](#_Toc40106123)

[1. Диаграмма вариантов использования 5](#_Toc40106124)

[2. Диаграмма классов 7](#_Toc40106125)

[3. Диаграммы взаимодействия 9](#_Toc40106126)

[4. Диаграммы состояний 15](#_Toc40106127)

[5. Диаграммы видов деятельности 21](#_Toc40106128)

[6. Диаграмма пакетов 24](#_Toc40106129)

[7. Диаграмма размещения 25](#_Toc40106130)

[ЗАКЛЮЧЕНИЕ 26](#_Toc40106131)

[СПИСОК ИСПОЛЬЗОВАНЫХ ИСТОЧНИКОВ 27](#_Toc40106132)

[Приложение А. Листинг сгенерированного кода 28](#_Toc40106133)

# ВВЕДЕНИЕ

В данной работе приведен проект построения программного обеспечения информационной системы. В качестве предметной области рассмотрена «АГЕНСТВО НЕДВИЖИМОСТИ», которая отражала бы работу реального предприятия и позволила бы упростить управление им. Проектирование производилось с помощью специализированного программного сервиса Creately, который представляет из себя инструмент для построения UML - схем. Данный сервис позволяет разработать UML модели, которые позволяют уменьшить временные, финансовые и другие издержки при разработки информационной системы. Этот язык не ограничивается моделированием программного обеспечения. Его также используют для моделирования бизнес-процессов, системного проектирования и отображения организационных структур предприятий.

К плюсом языка UML можно выделить:

1. UML объектно-ориентированный язык, в результате чего методы описания результатов анализа и проектирования семантически близки к методам программирования на современных объектно-ориентированных языках;
2. UML позволяет описать информационную систему практически со всех возможных точек зрения, включая разные аспекты поведения системы;
3. Диаграммы UML сравнительно просты для чтения после достаточно быстрого ознакомления с синтаксисом языка;
4. Сокращение числа возможных ошибок таких как: несогласованные параметры подпрограмм, несогласованное изменение атрибутов;
5. Повторное использование. Предполагается возможность многократного использования уже существующего проекта или его частей в новом проекте;
6. UML расширяет и позволяет вводить собственные текстовые и графические стереотипы, что способствует его применению не только в сфере программной инженерии;
7. UML получил широкое распространение и динамично развивается.

## Проектирование программного обеспечения информационной системы «Агентство недвижимости»

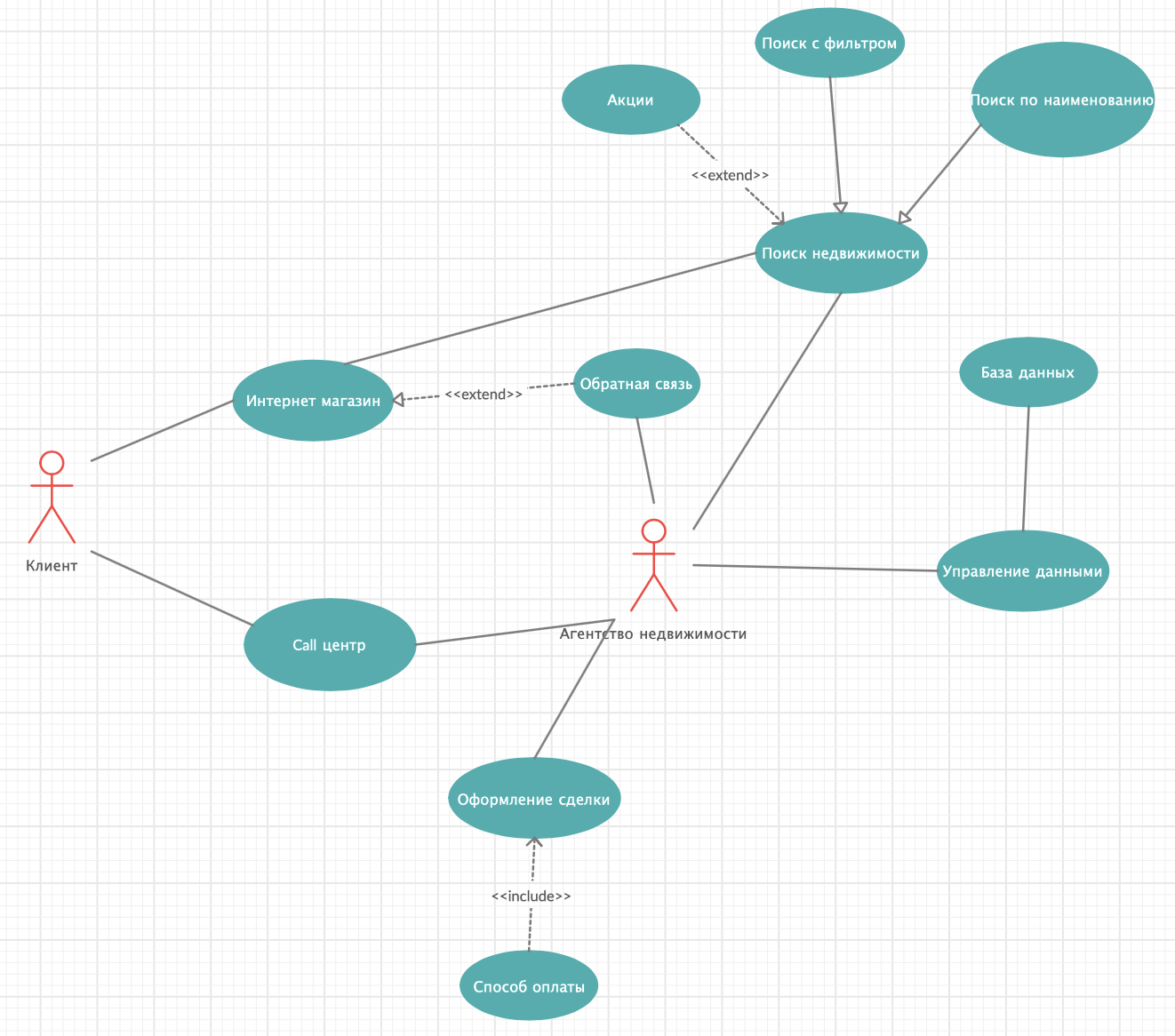
Разрабатываемая информационная система предназначена для осуществления работы основных процессов внутри магазина и его взаимодействия с внешними системами.

Система выполняет такие функции, как:

1. выбор недвижимости и оплата клиентом;
2. просмотр недвижимости;
3. просмотр недвижимости с применением различных фильтром поиска, акций и поиск по наименованию позиции в БД;
4. заключение договора между агентством недвижимости и клиентом;
5. быстрей online ответ клиенты через call-центр;

## Диаграмма вариантов использования

В диаграмме вариантов использования (рисунок 1) отражено функционирование информационной системы «Агентство недвижимости».



1. Диаграмма вариантов использования

#### *Расчет количественной оценки информационной наполненности диаграммы вариантов использования*

Для оценки диаграмм UML используется методика, позволяющая рассчитать количественный коэффициент, описывающий такие параметры диаграммы как наглядность, удобство восприятия, степень информативности (или загруженности информацией). Для расчета данного коэффициента используется формула:

где Sobj-оценка элемента на диаграмме, Slink- оценка связей, Оbj- кол-во объектов на диаграмме, Tobj –количество типов объектов, Tlink- количество типов связи.

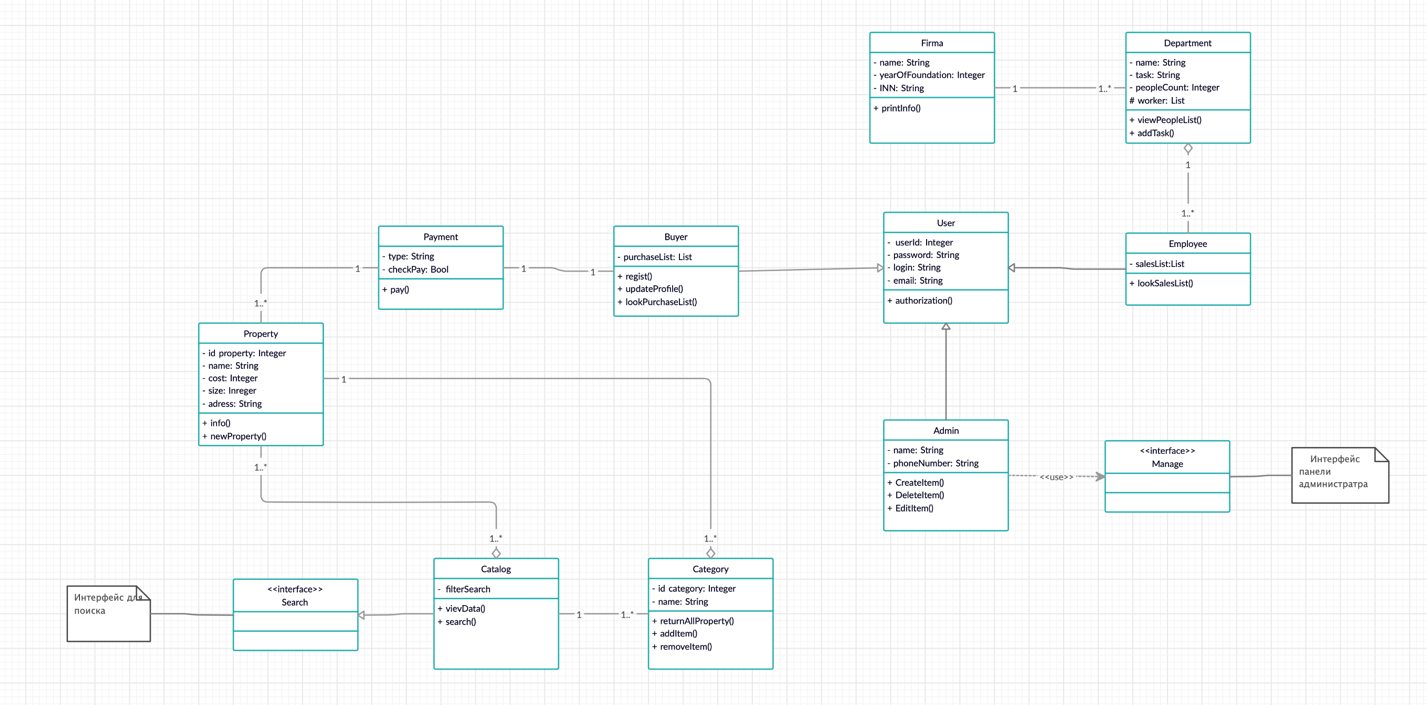
Для данной диаграммы оценка вычислена по формуле 1 и равна:

## Диаграмма классов

Диаграмма классов служит для представления статической структуры модели системы в терминологии классов объектно-ориентированного программирования. Диаграмма классов может отражать, в частности, различные взаимосвязи между отдельными сущностями предметной области, такими как объекты и подсистемы, а также описывает их внутреннюю структуру и типы отношений.

#### *Описание диаграммы классов*

Диаграмма классов (рисунок 2) служит для наглядного взаимодействия классов системы.



1. Диаграмма классов

#### *Расчет количественной оценки информационной наполненности диаграммы классов*

Расчёт количественной оценки информационной наполненности диаграммы классов оценивается по формулам 1-2.

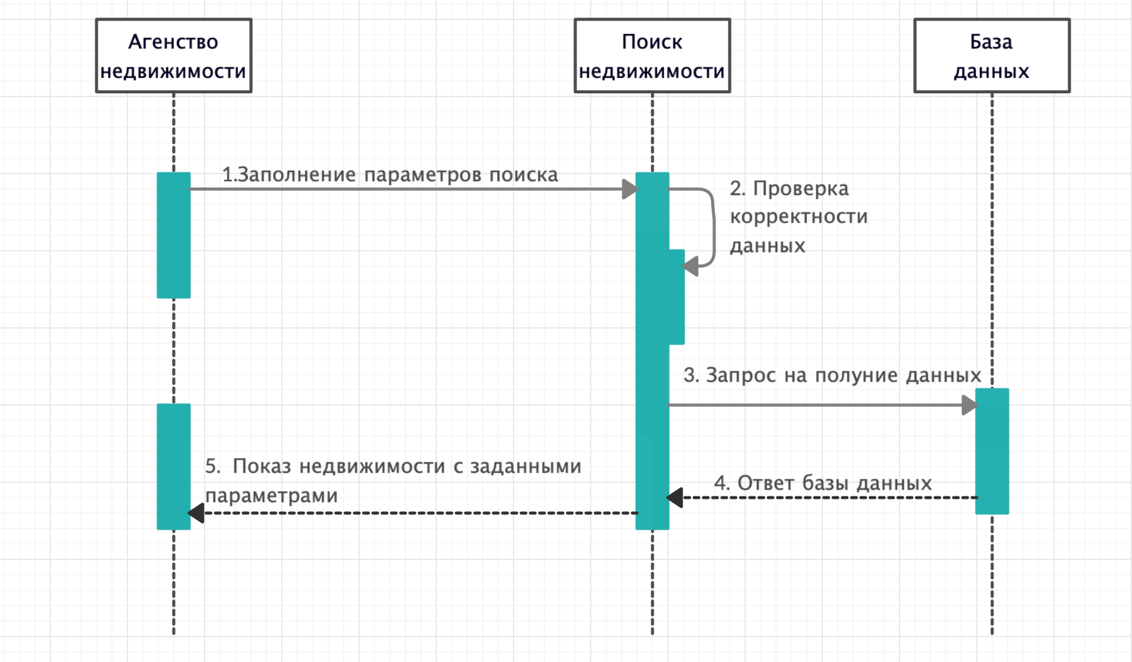
Оценки классов:

Оценка диаграммы:

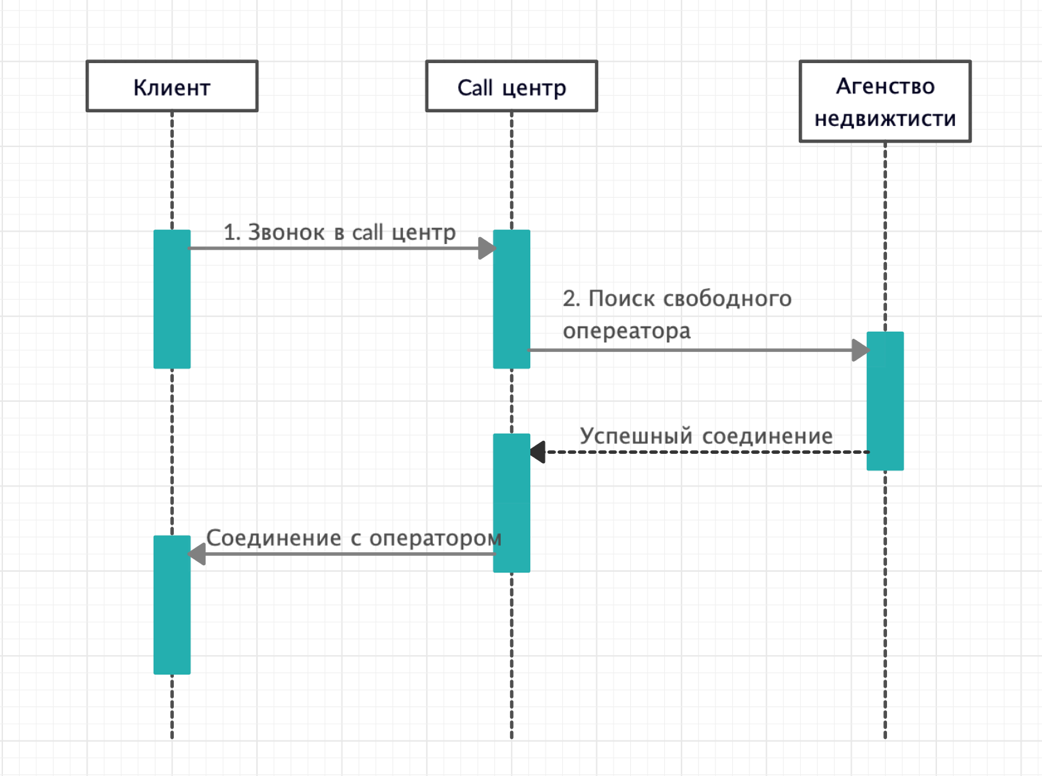
## Диаграммы взаимодействия

*Диаграмма последовательности* – это диаграмма, отображающая взаимодействия между объектами в рамках текущего сценария, упорядоченные во времени.

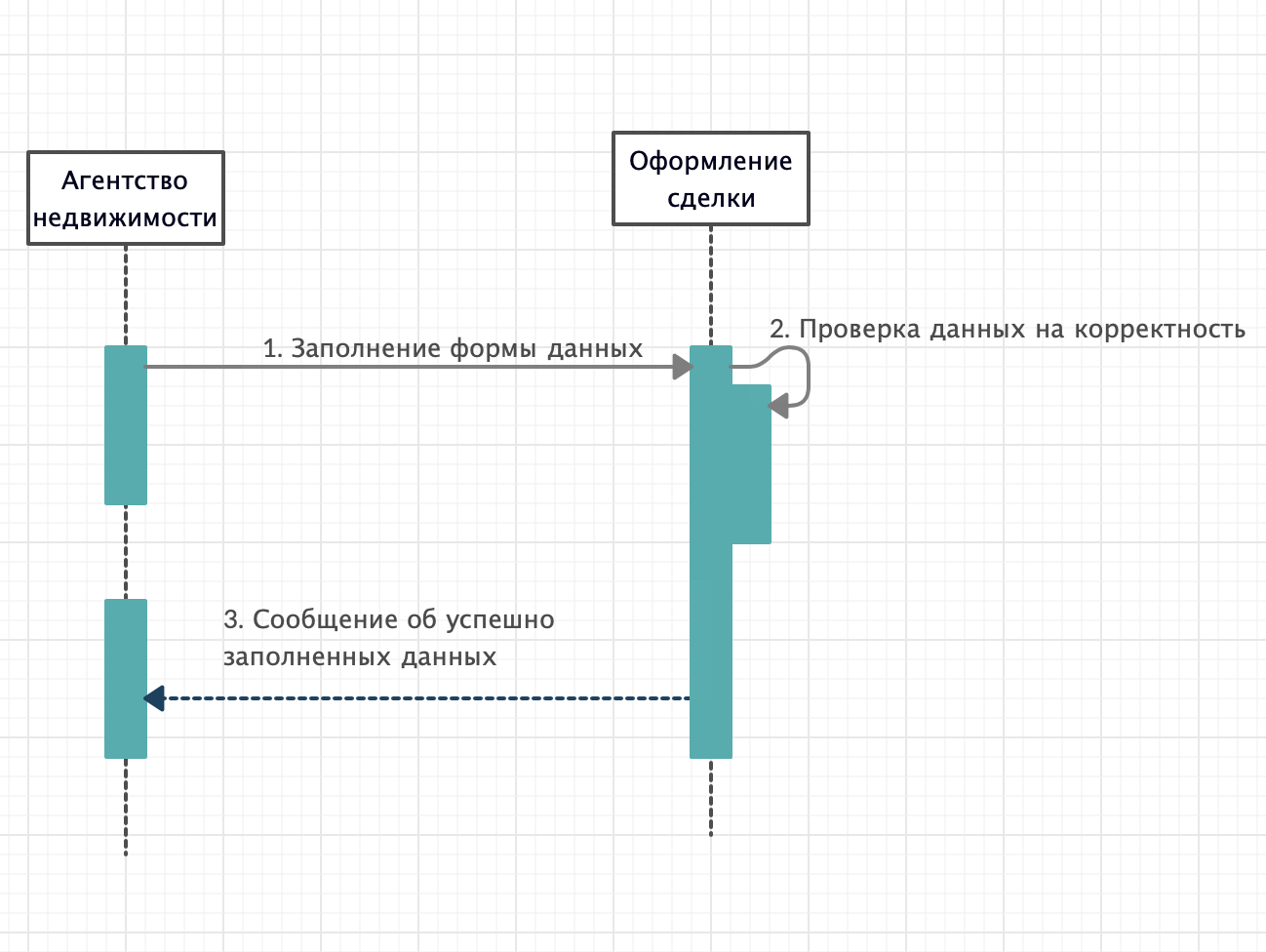
#### *Описания диаграмм последовательности*



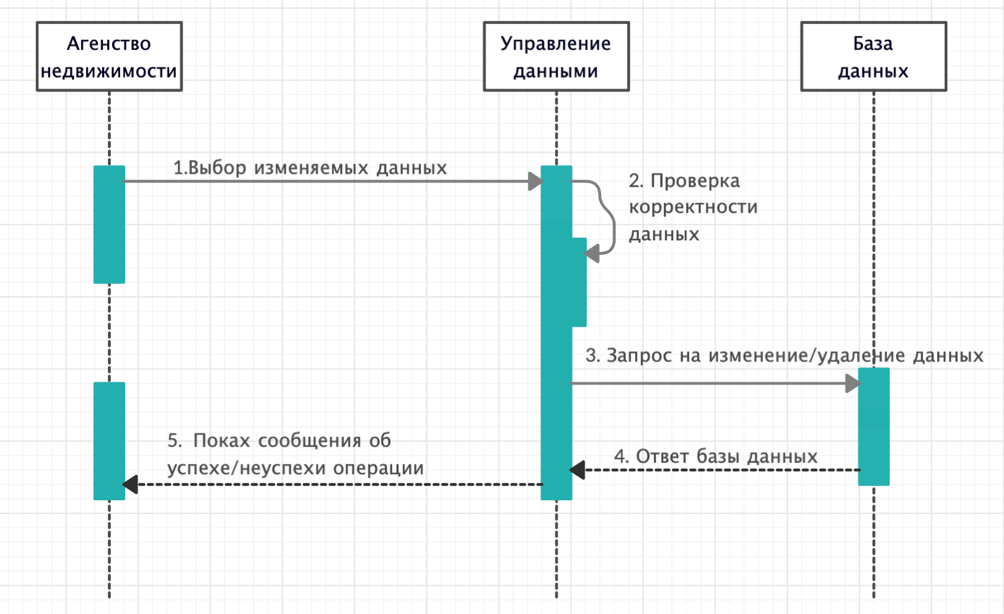
1. Диаграмма последовательности «Поиск недвижимости»



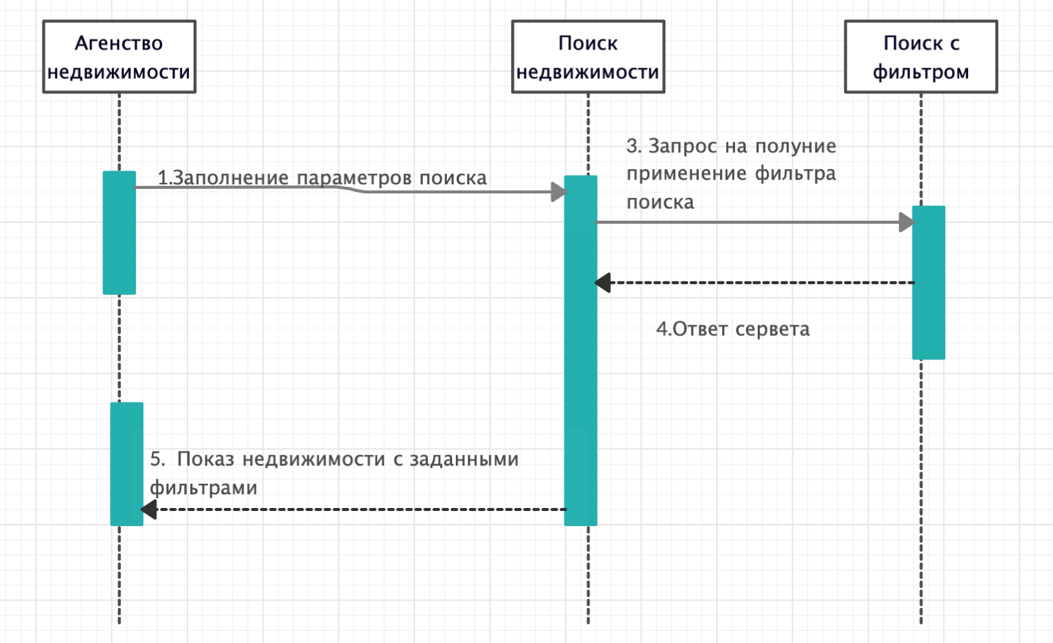
1. Диаграмма последовательности «Call центр»



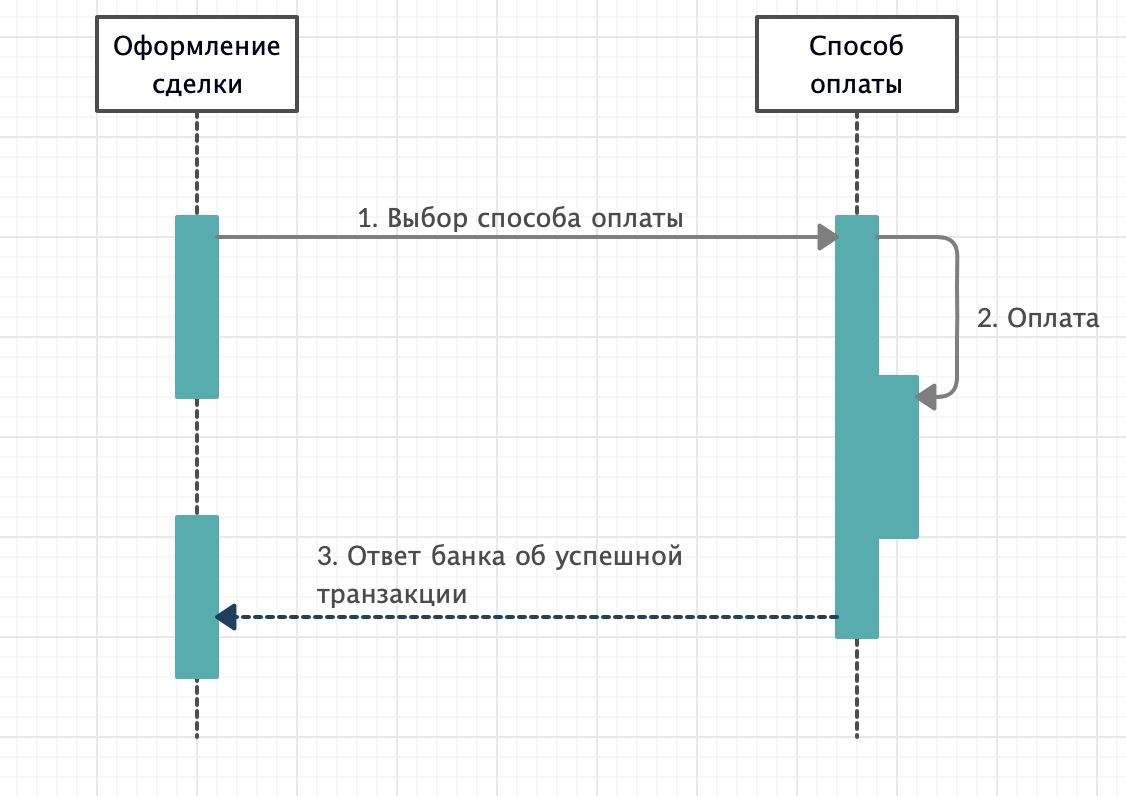
1. Диаграмма последовательности «Оформление сделки»



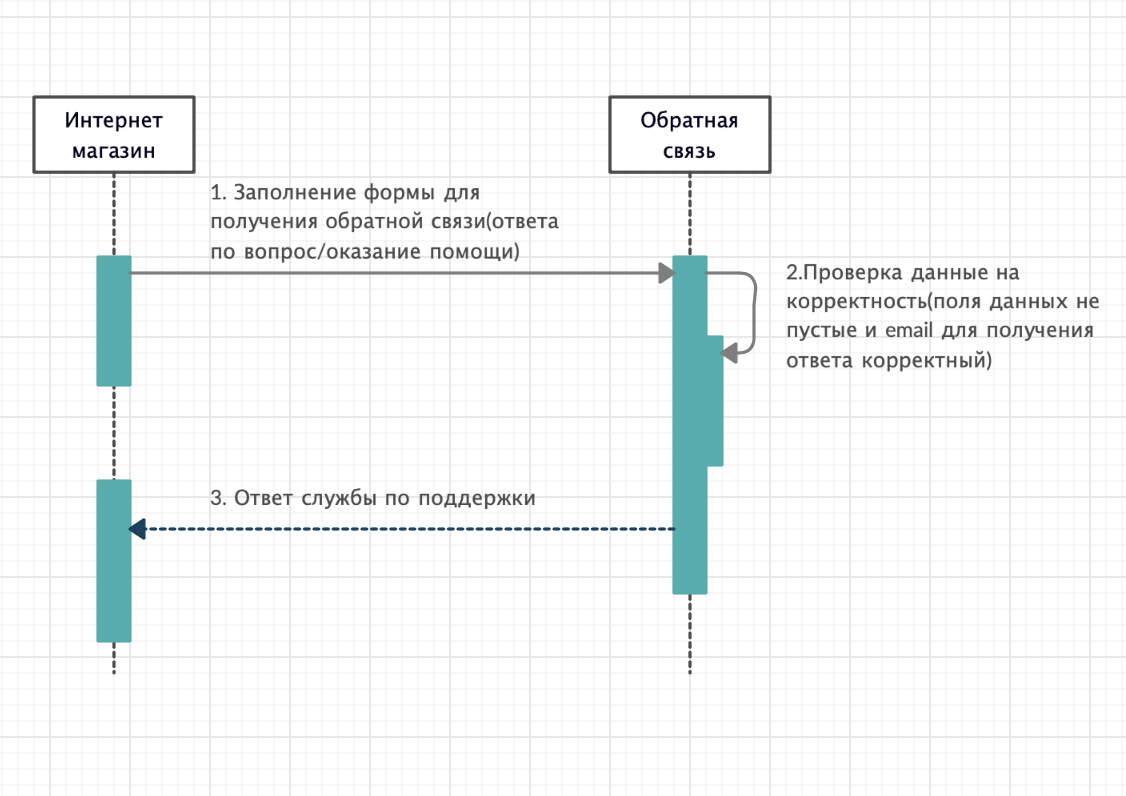
1. Диаграмма последовательности «Управление данными»



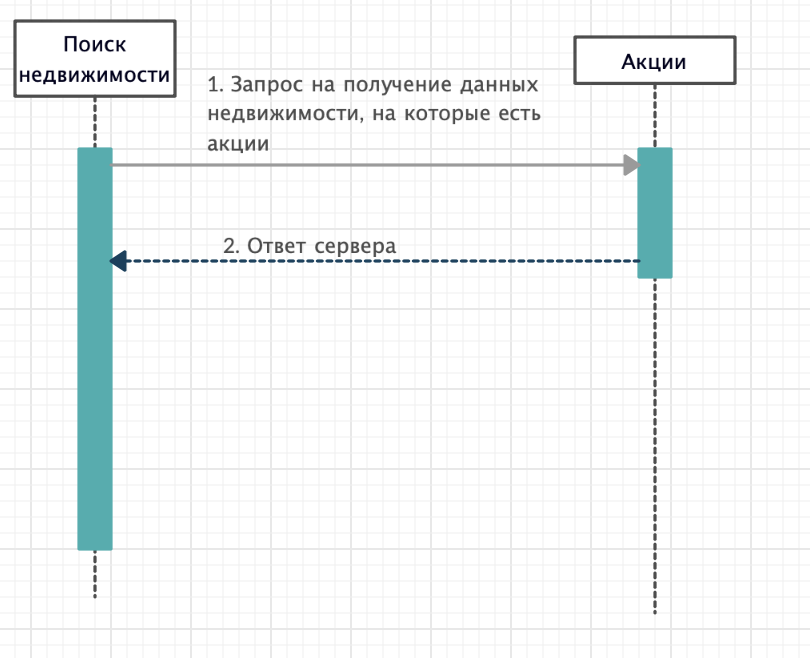
1. Диаграмма последовательности «Поиск недвижимости с фильтром»



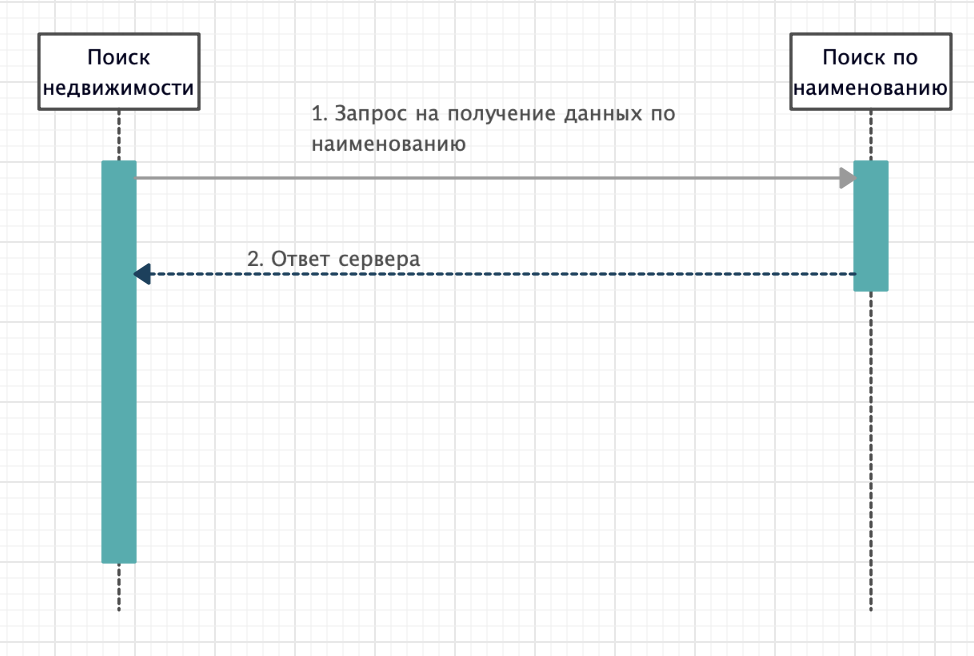
1. Диаграмма последовательности «Способ оплаты»



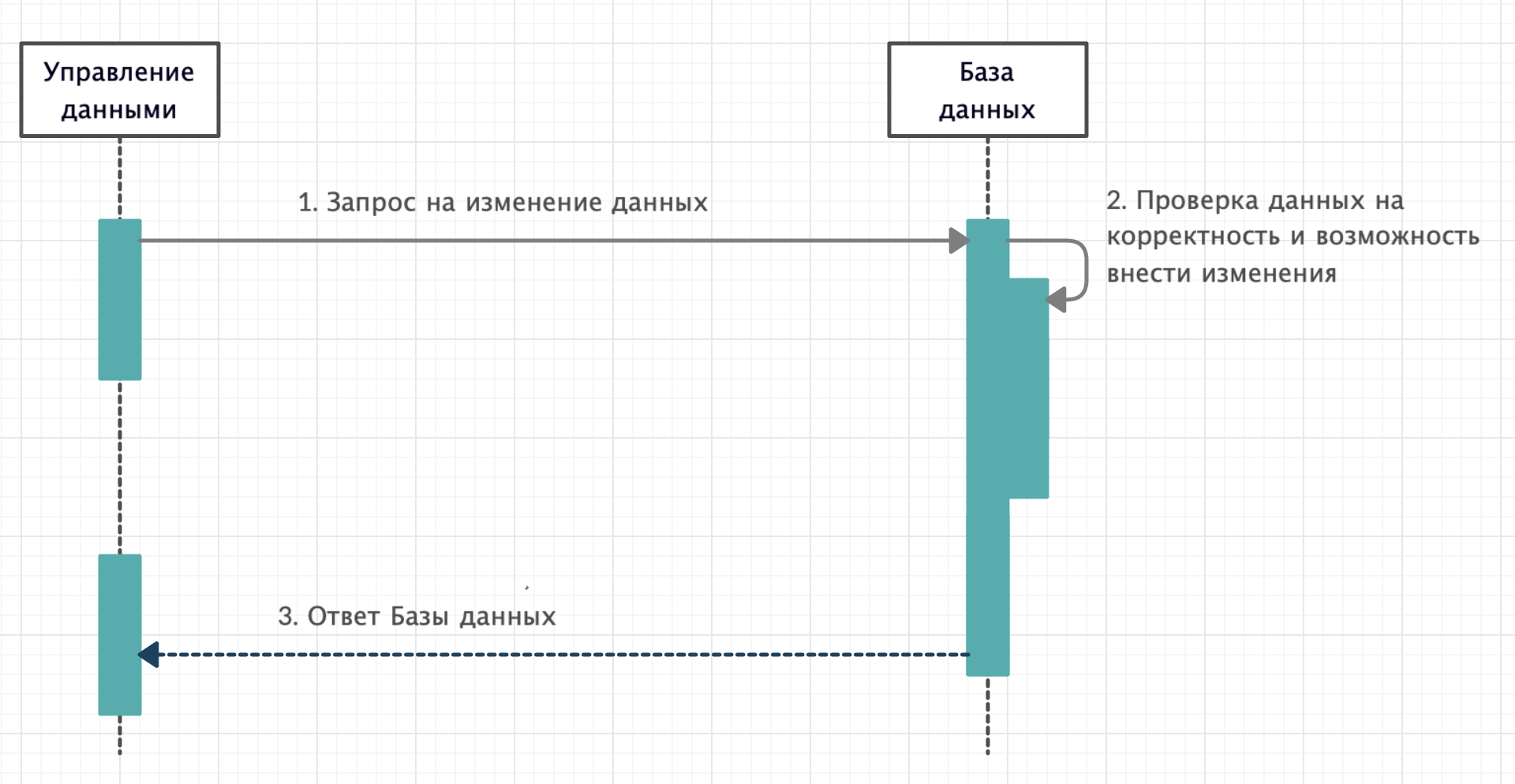
1. Диаграмма последовательности «Обратная связь»



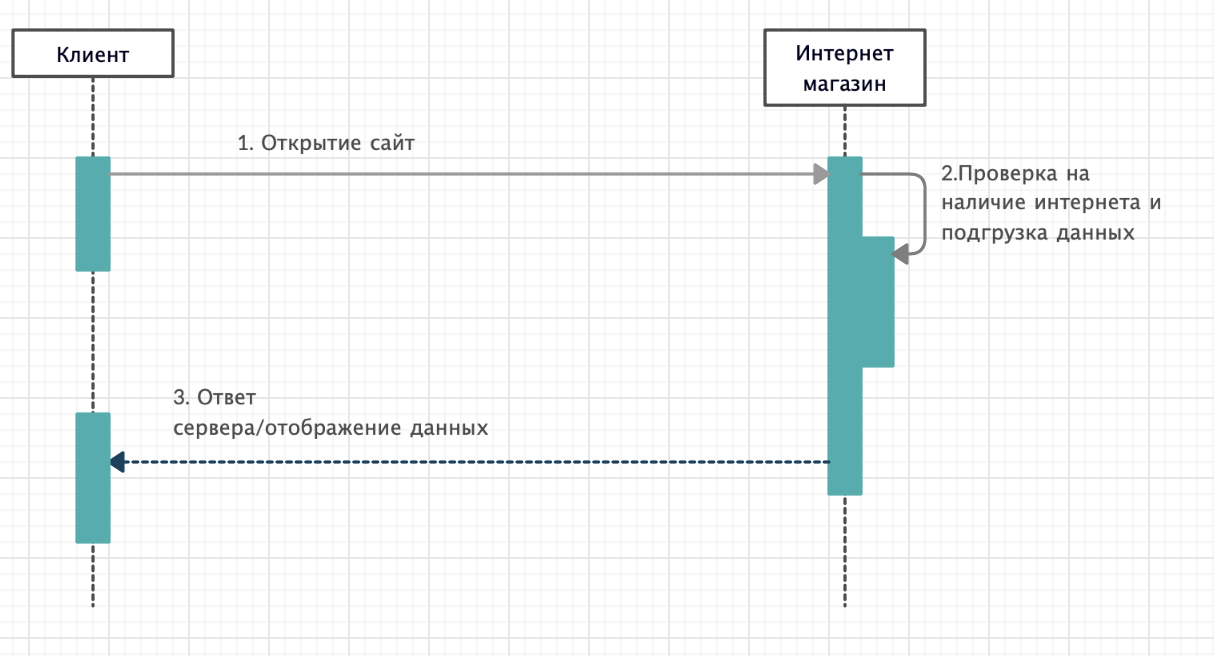
1. Диаграмма последовательности «Акции»



1. Диаграмма последовательности «Поиск по наименованию»



1. Диаграмма последовательности «База данных»



1. Диаграмма последовательности «Интернет магазин»

#### *Расчет количественной оценки информационной наполненности диаграмм последовательности*

1. Оценка диаграмм последовательности

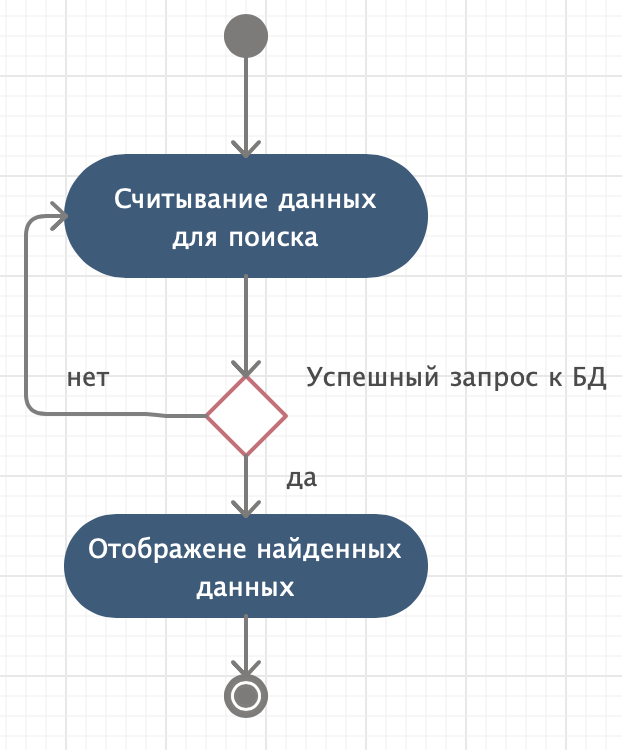
|  |  |
| --- | --- |
| Диаграмма последовательности | Оценка |
| Поиск недвижимости | 3.7 |
| Call центр | 3.5 |
| Оформление сделки | 2.95 |
| Управление данными | 3.7 |
| Поиск недвижимости с фильтром | 3.5 |
| Способ оплаты | 2.95 |
| Обратная связь | 2.95 |
| Акции | 2.73 |
| Поиск по наименованию | 2.73 |
| База данных | 2.95 |
| Интернет магазин | 2.95 |

### 

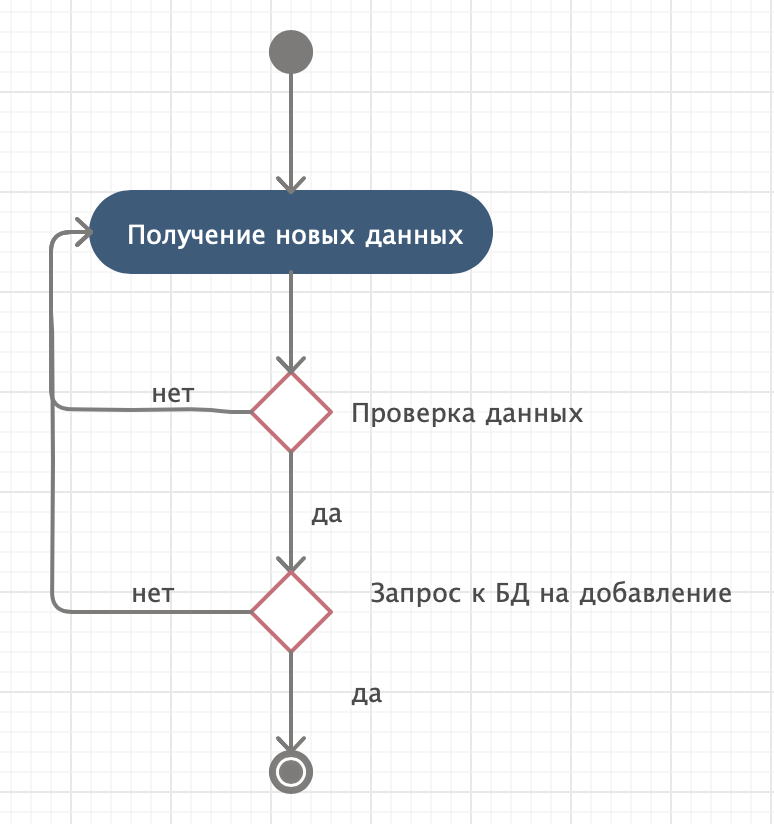
## Диаграммы состояний

*Диаграммы состояний* – это один из пяти видов диаграмм UML, предназначенных для моделирования динамических аспектов поведения систем. Диаграмма состояний показывает конечный автомат. И диаграммы деятельности, и диаграммы состояний подходят для моделирования жизненного цикла объекта. Однако в то время, как диаграмма видов деятельности демонстрирует поток управления от одной деятельности к другой через множество объектов, диаграмма состояний отображает поток управления от состояния к состоянию внутри отдельного объекта.

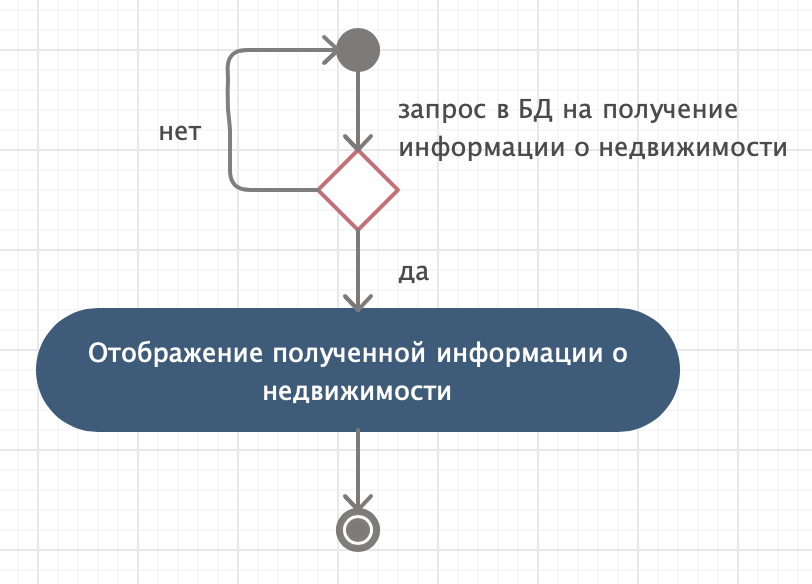
#### *Описание диаграмм состояний*



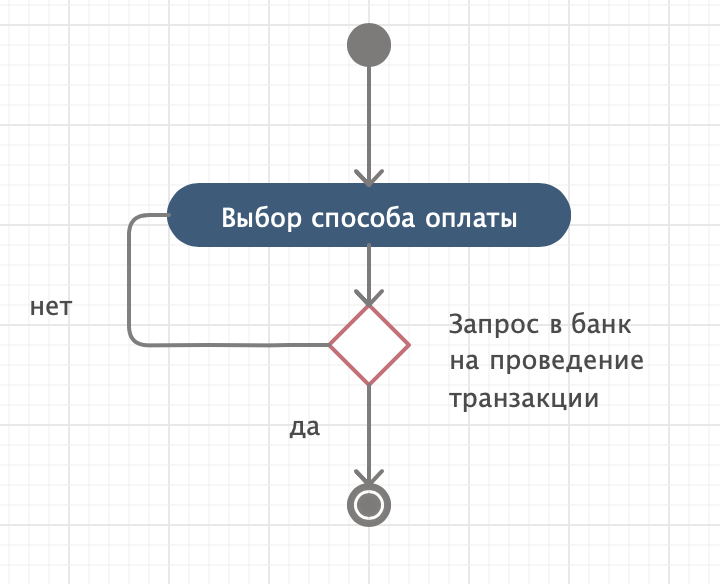
1. Диаграмма состояния Catalog



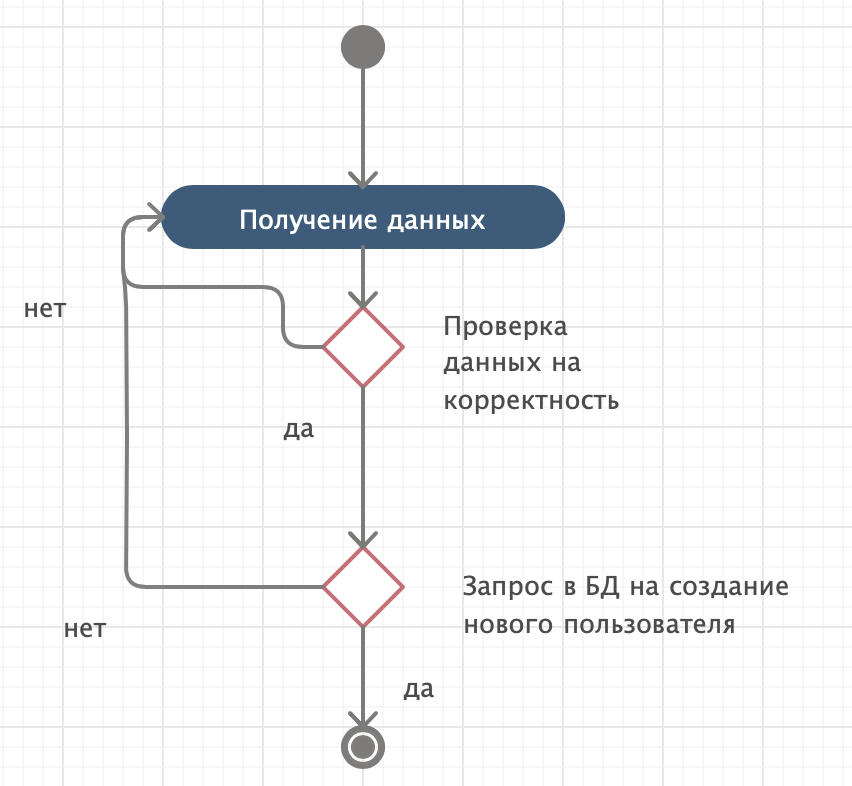
1. Диаграмма состояния



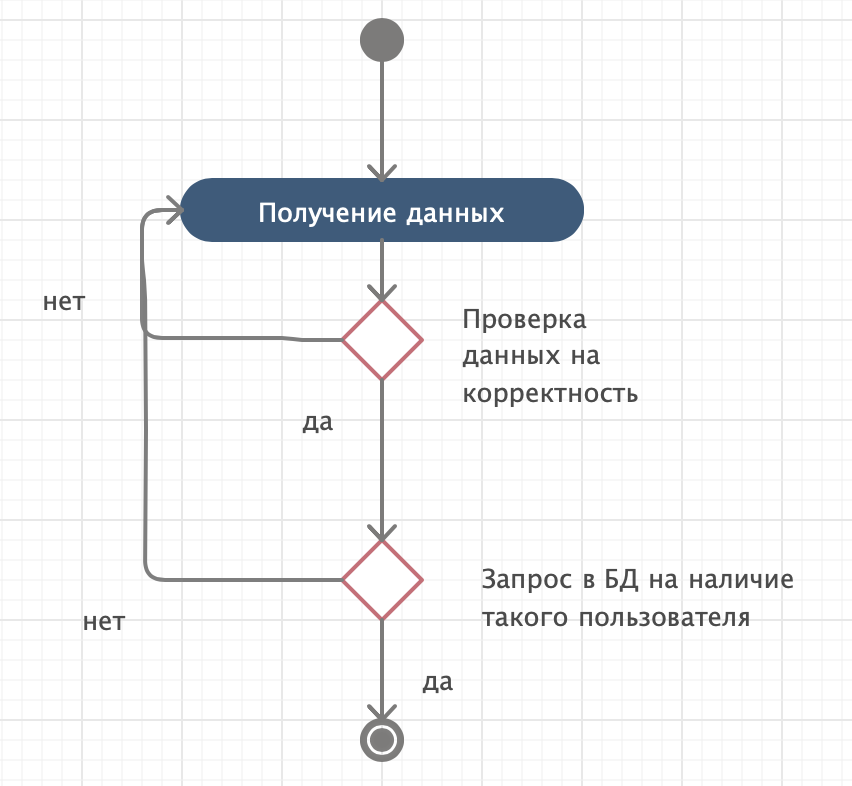
1. Диаграмма состояния



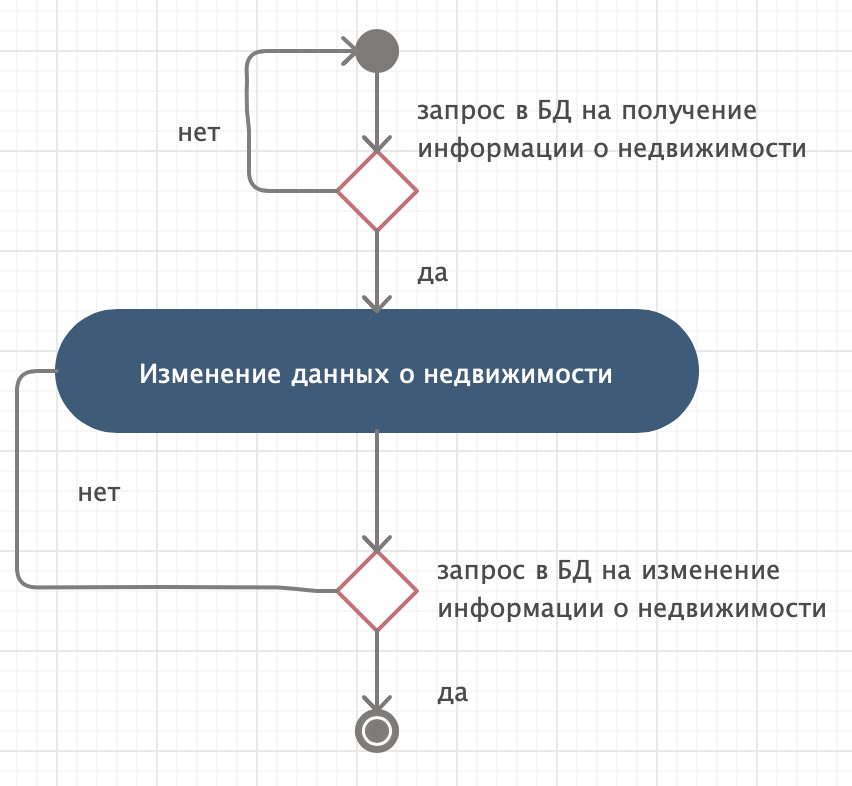
1. Диаграмма состояния



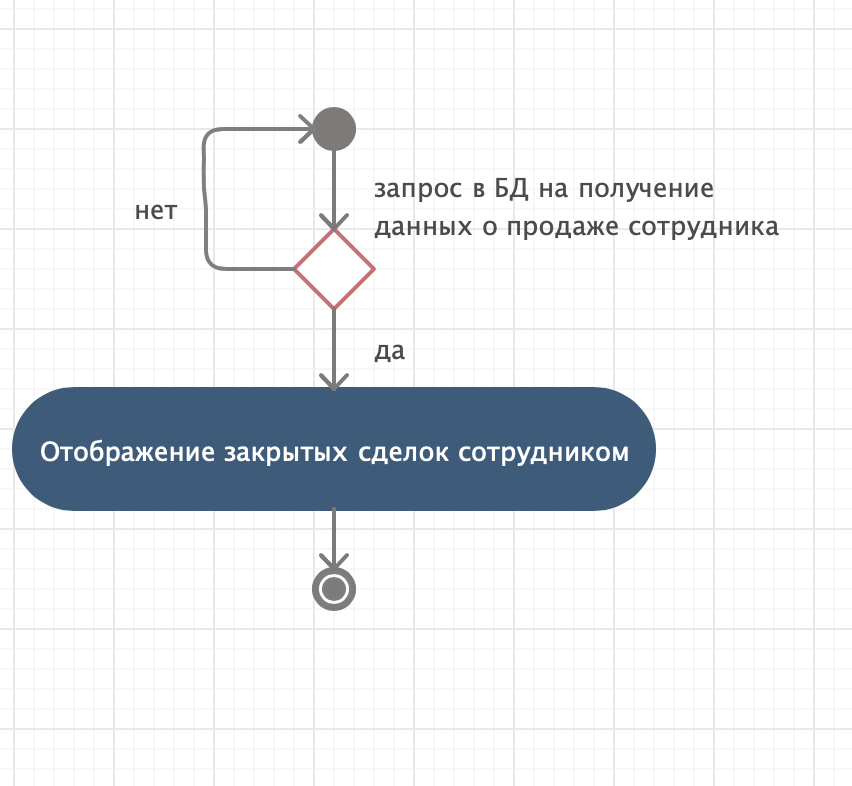
1. Диаграмма состояния



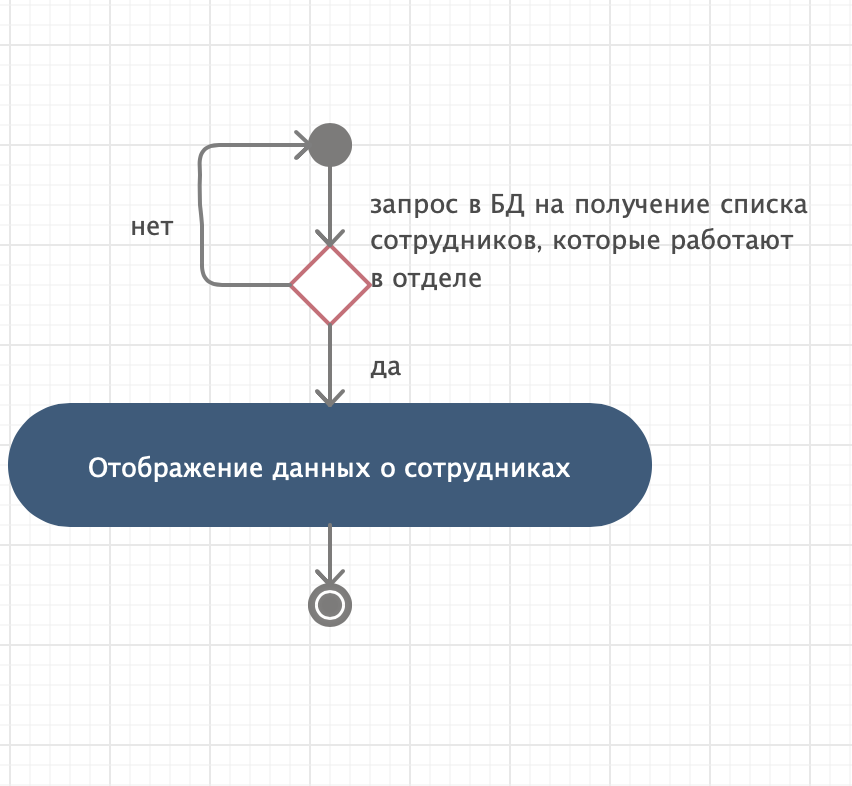
1. Диаграмма состояния



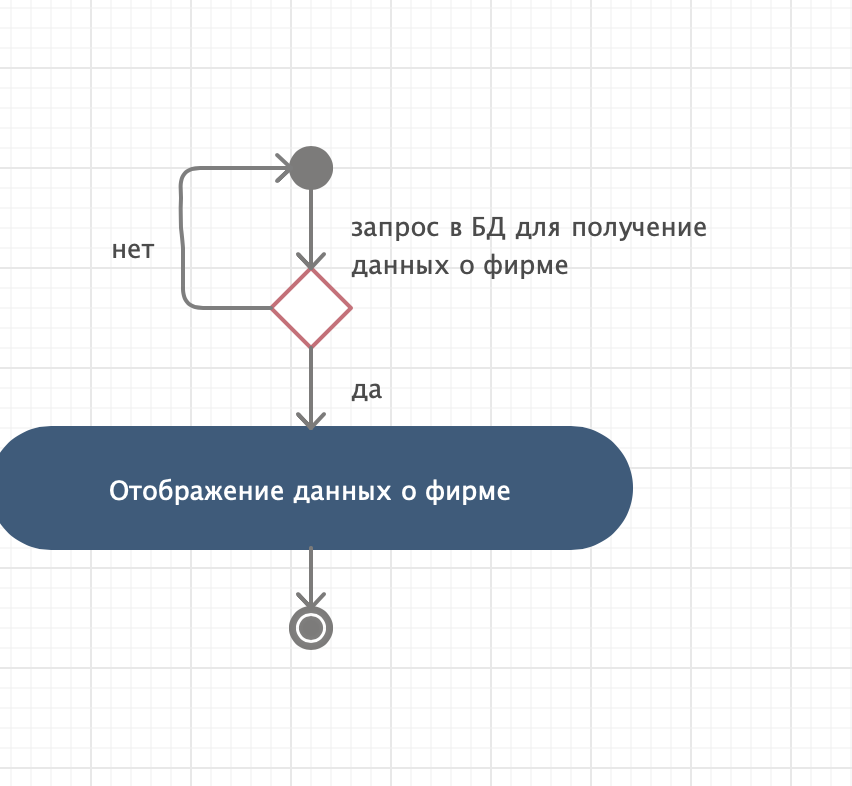
1. Диаграмма состояния



1. Диаграмма состояния



1. Диаграмма состояния



1. Диаграмма состояния

#### *Расчет количественной оценки информационной наполненности диаграмм состояний*

1. Оценка диаграмм состояний

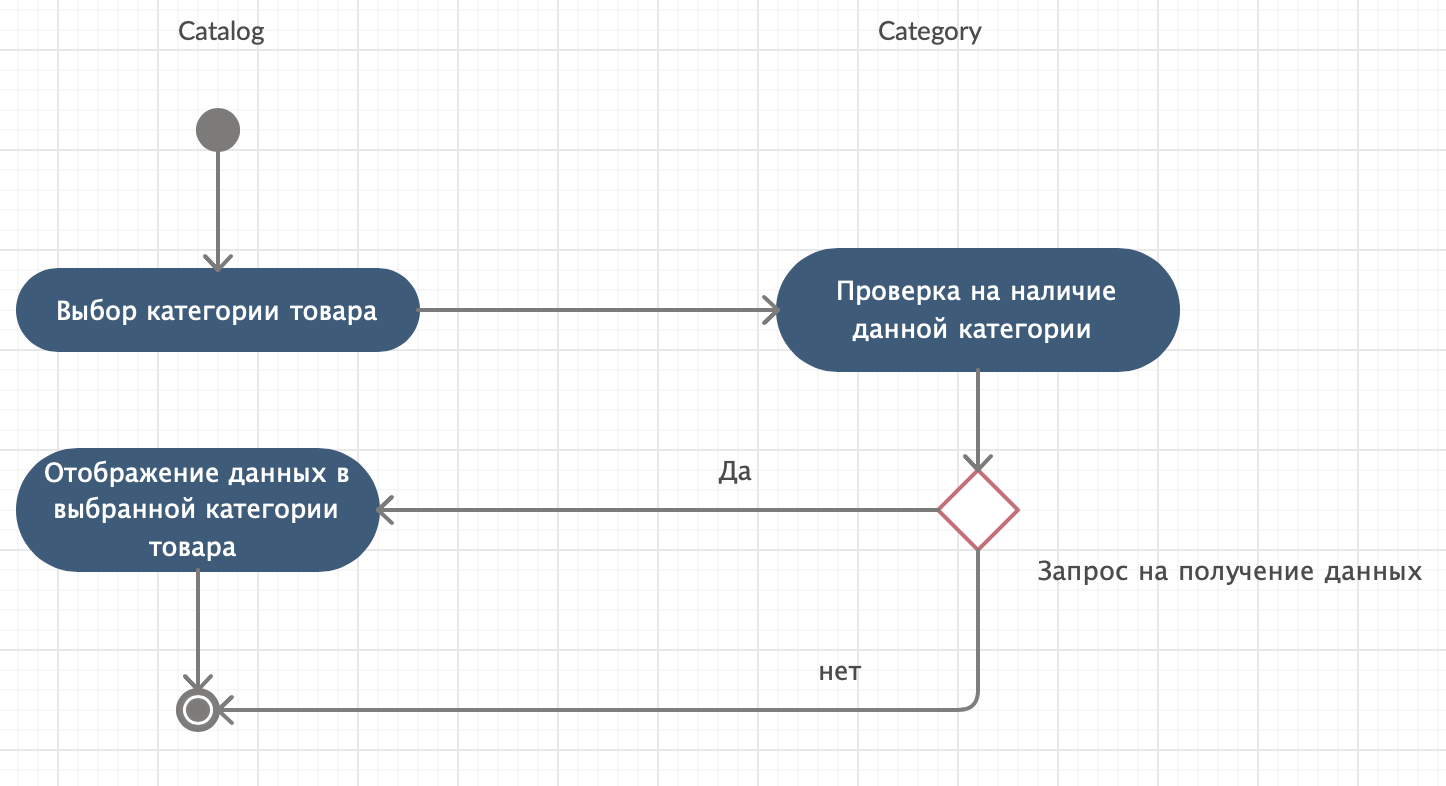
|  |  |
| --- | --- |
| Диаграмма | Оценка |
| Catalog | 2.3 |
|  | 1.8 |
|  | 1.7 |
|  | 1.7 |
|  | 1.8 |
|  | 1.8 |
|  | 1.8 |
|  | 1.7 |
|  | 1.7 |
|  | 1.7 |

### 

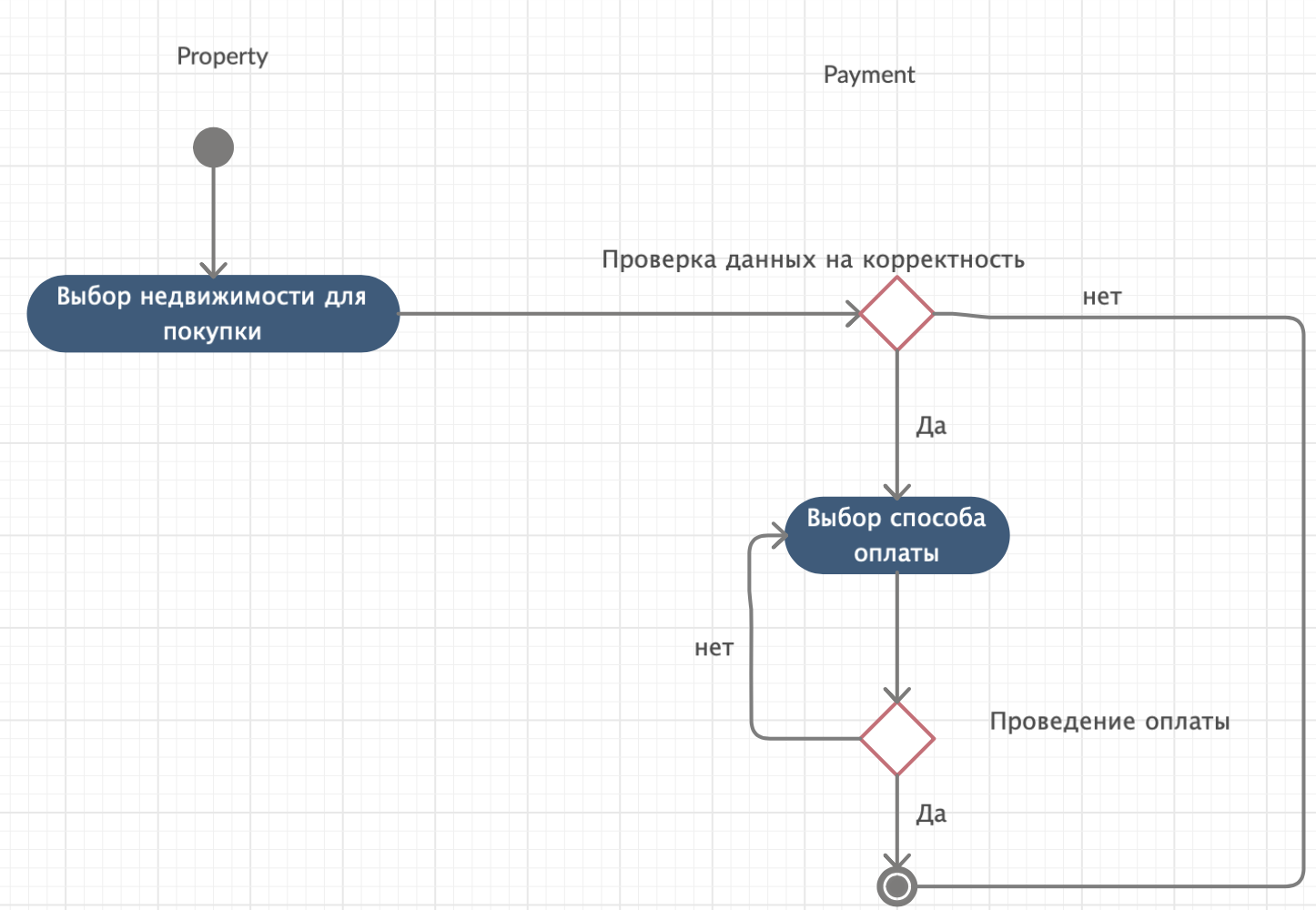
## Диаграммы видов деятельности

*Диаграммы видов деятельности* – это один из пяти видов диаграмм, применяемых в UML для моделирования динамических аспектов систем. По сути, диаграмма деятельности представляет собой блок-схему, которая показывает, как поток управления переходит от одной деятельности к другой. В отличие от традиционной блок-схемы диаграмма деятельности показывает параллелизм так же хорошо, как и ветвление потока управления.

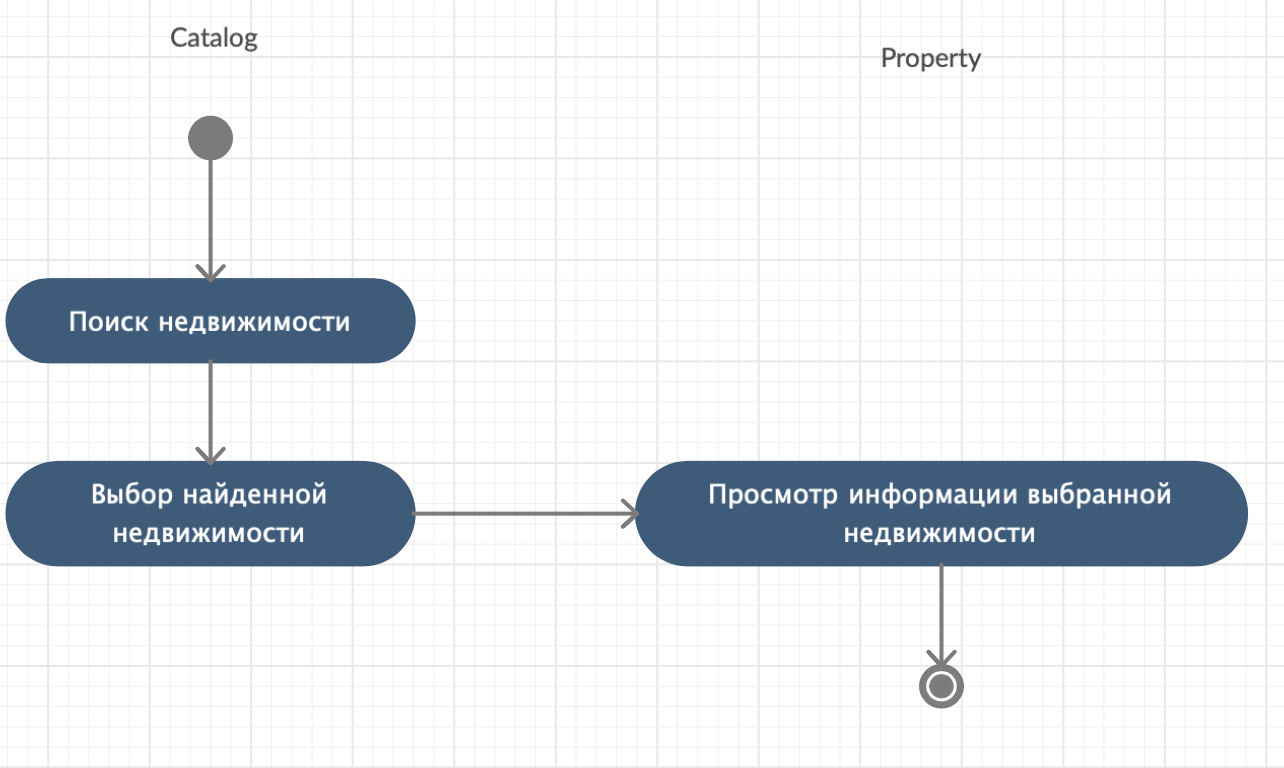
#### *Описания диаграмм видов деятельности*



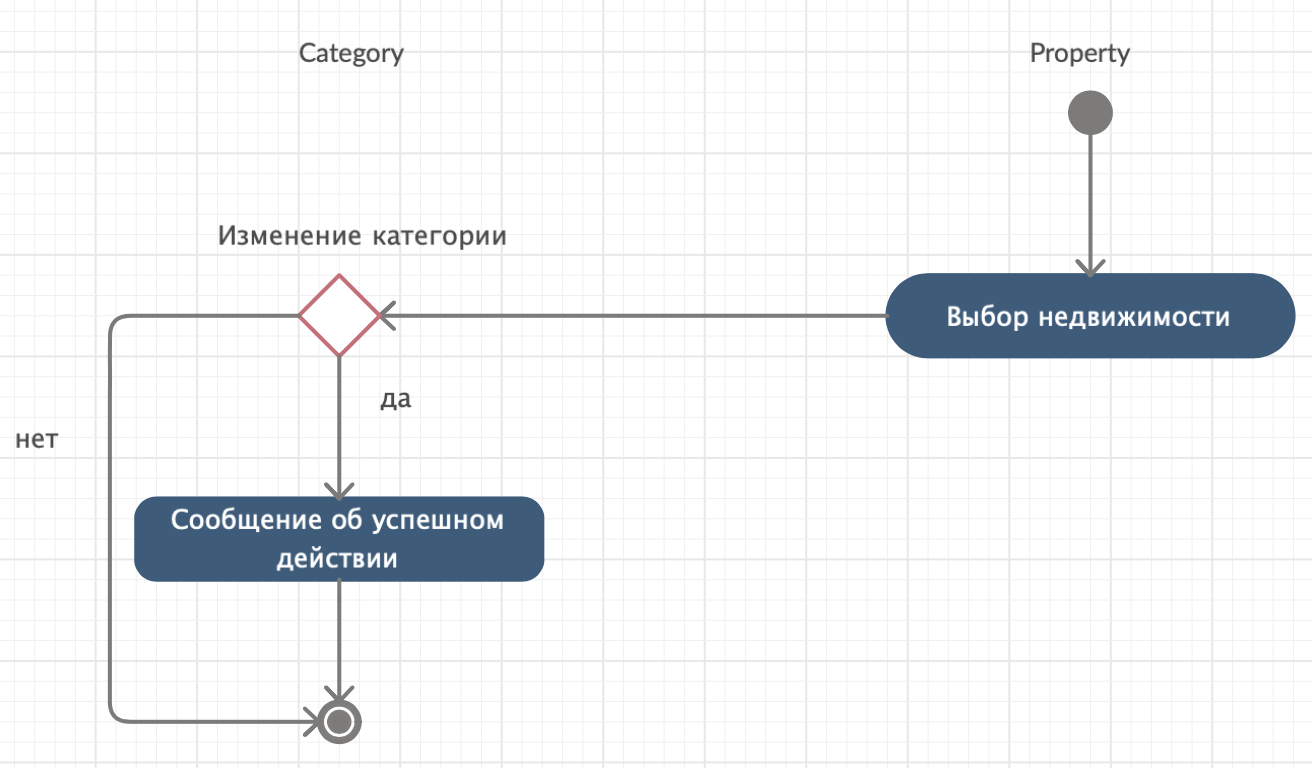
1. Диаграмма видов деятельности Поиск



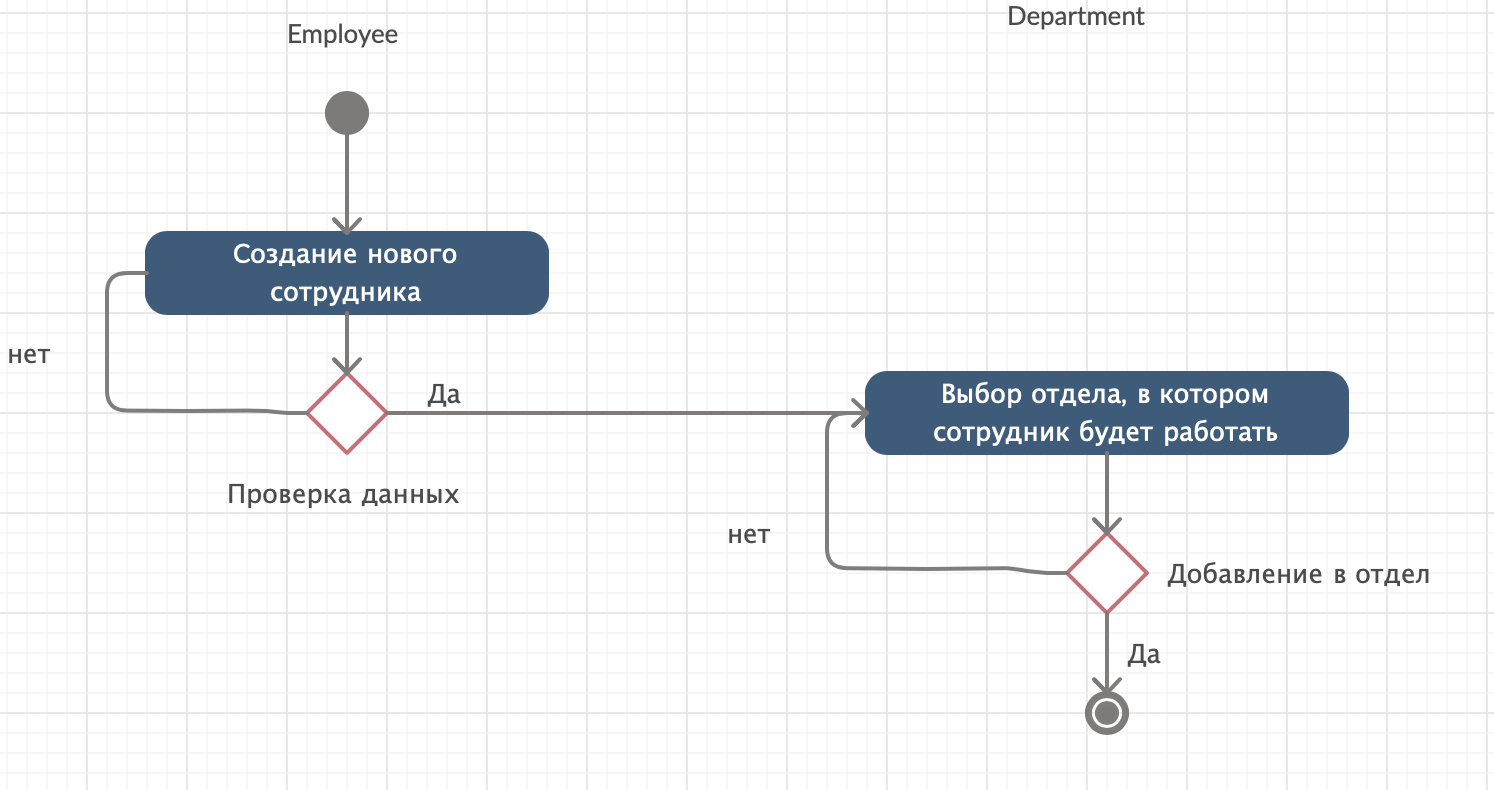
1. Диаграмма видов деятельности Оплата



1. Диаграмма видов деятельности Просмотр недвижимости



1. Диаграмма видов деятельности Изменение категории недвижимости



1. Диаграмма видов деятельности Создание сотрудника

#### *Расчет количественной оценки информационной наполненности диаграмм видов деятельности*

1. Оценка диаграмм видов деятельности

|  |  |
| --- | --- |
| Диаграмма последовательности | Оценка |
| Поиск по категориям | 2.9 |
| Оплата | 2.2 |
| Просмотр недвижимости | 2.8 |
| Изменение категории недвижимости | 2.3 |
| Создание сотрудника | 3.2 |

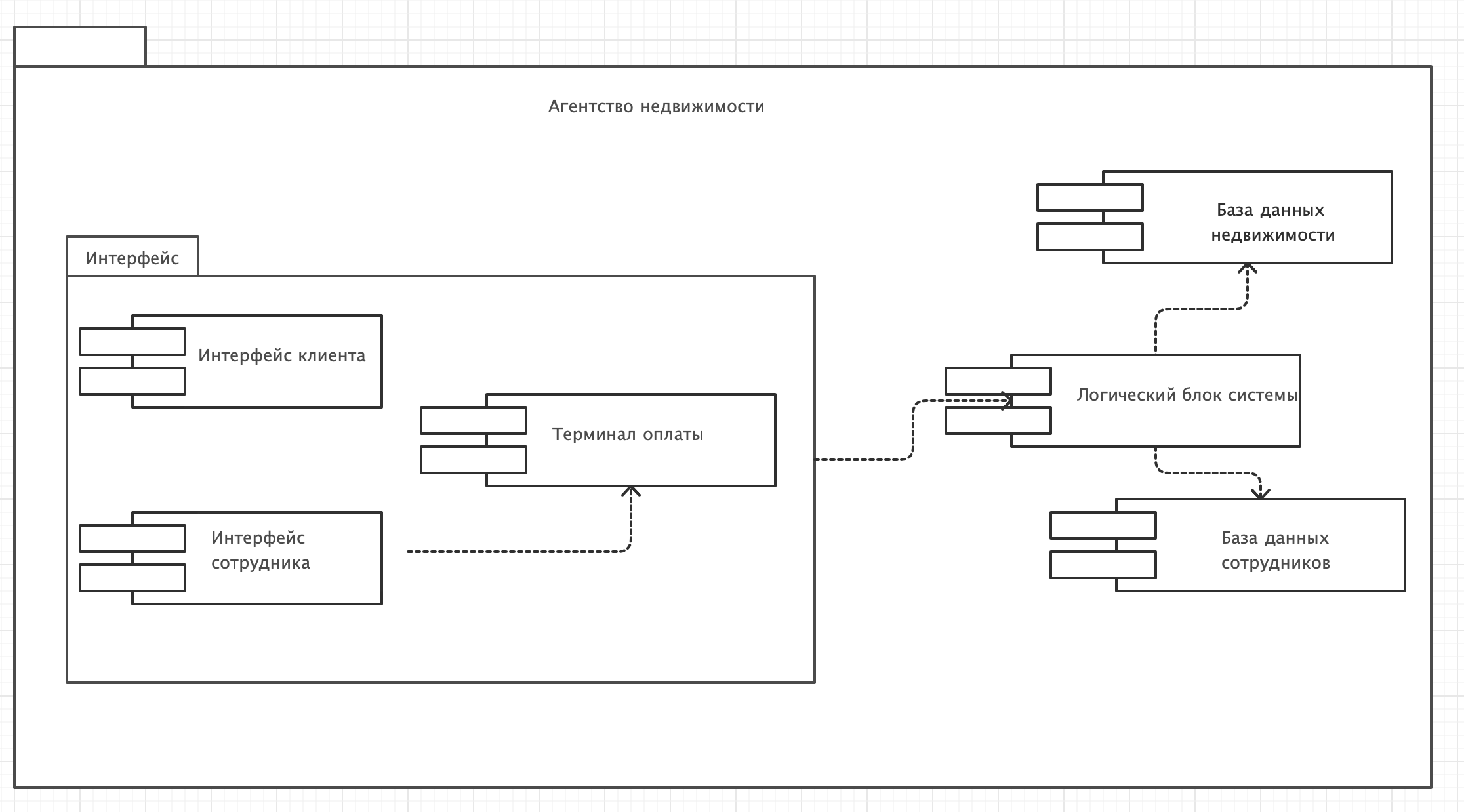
### 

## Диаграмма пакетов

*Диаграмма пакетов* – это диаграмма, представляющая логическую группировку составляющих информационную систему процессов и объектов. Данная диаграмма предназначена для обеспечения удобства разработки, так как позволяет группировать объекты любым способов. При этом один объект или пакет может принадлежать нескольким родительским пакетам.

#### *Описание диаграммы пакетов*

Диаграмма пакетов показывает, как хранятся, располагаются и взаимодействую между собой объекты информационной системы.



1. Диаграмма пакетов

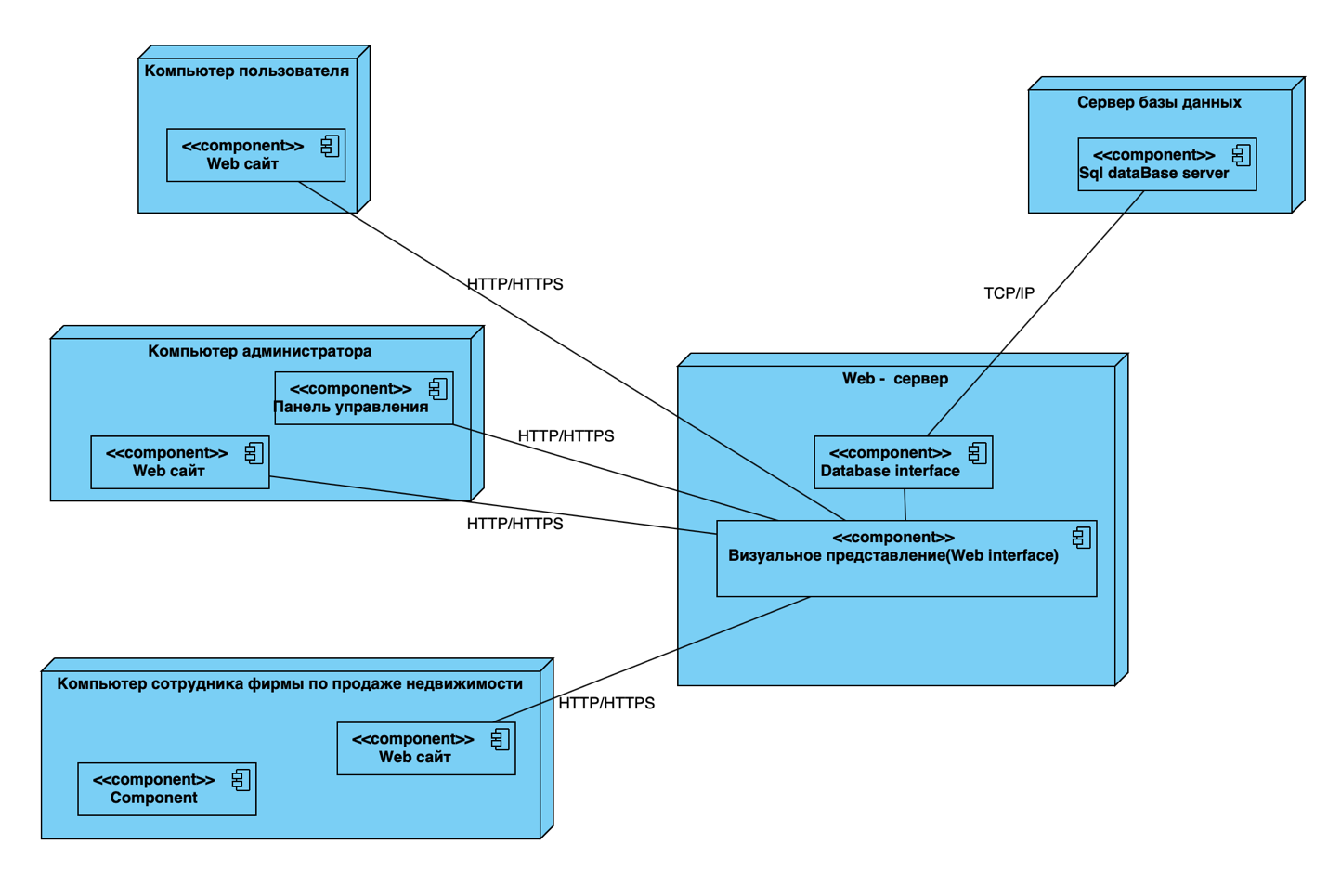
#### *Расчет количественной оценки информационной наполненности диаграммы пакетов*

## Диаграмма размещения

*Диаграмма размещения* – это диаграмма, предназначенная для моделирования работающих узлов (аппаратных средств), и компонентов информационной системы, развернутых на них.

#### *Описание диаграммы размещения*

Диаграмма размещения представлена на рисунке 40.



1. Диаграмма размещения

#### *Расчет количественной оценки информационной наполненности диаграммы размещения*

# ЗАКЛЮЧЕНИЕ

В данной курсовой работе была спроектирована информационная система «Агентство недвижимости» с помощью языка UML. Информационная система включает в себя: диаграмму вариантов использования (Use Case Diagram), диаграмму классов (Class Diagram), диаграммы последовательности (Sequence Diagram), диаграммы состояний (Statechart Diagram), диаграммы видов деятельности (Activity Diagram), диаграмму пакетов (Package Diagram) и диаграмму развертывания (Deployment Diagram).

Спроектированная система является наглядной основой для дальнейших модификаций, результатом которых станет возможность перехода к разработке полноценной информационной системы.

В результате генерации кода для каждого класс получены файлы с расширением .cpp и .h.

# СПИСОК ИСПОЛЬЗОВАНЫХ ИСТОЧНИКОВ

1. Пятлина Е.О. Учебное пособие по КР ООПрИС / Пятлина Е.О. , Ключарёв А.А., Павлов Е.В., 2020 - 50c.
2. Алистер Коберн. Современные методы описания функциональных требований к системам / Алистер Коберн. – Лори, 2017. – 288 с.
3. Леоненков А.В. Нотация и семантика языка UML / Леоненков А.В. – М.: Интуит, 2016. — 205 c.
4. Буч Грэди Введение в UML от создателей языка / Грэди Буч, Джеймс Рамбо, Айвар Якобсон: пер. с англ. – ДМК Пресс, 2015 – 496 с.: ил.
5. Лешек А. Мацяшек, Анализ и проектирование информационных систем с помощью UML 2.0 / Лешек А. Мацяшек. – Вильямс, 2016. – 816 c.

# Приложение А. Листинг сгенерированного кода

Admin.cpp

#include "Admin.h"

Admin::Admin(){}

initAttributes(){}

Admin::~Admin(){}

void Admin::initAttributes(){}

Admin.h

#ifndef ADMIN\_H

#define ADMIN\_H

#include "User.h"

#include <string>

class Admin : public User{

public:

Admin();

virtual ~Admin();

void createItem(){}

void DeleteItem(){}

void EditItem(){}

protected:

private:

std::string name;

std::string phoneNumber;

void setName(std::string new\_var){

name = new\_var;

}

std::string getName(){

return name;

}

void setPhoneNumber(std::string new\_var){

phoneNumber = new\_var;

}

std::string getPhoneNumber() {

return phoneNumber;

}

void initAttributes(){}

#endif // ADMIN\_H

Buyer.cpp

#include "Buyer.h"

Buyer::Buyer(){

initAttributes();

}

Buyer::~Buyer(){}

void Buyer::initAttributes(){}

Buyer.h

#ifndef BUYER\_H

#define BUYER\_H

#include "User.h"

#include <string>

#include <vector>

class Buyer : public User{

public:

Buyer();

virtual ~Buyer();

void regist(){}

void updateProfile(){}

void lookPurchaseList(){}

protected:

private:

std::string purchaseList;

void setPurchaseList(std::string new\_var){

purchaseList = new\_var;

}

std::string getPurchaseList(){

return purchaseList;

}

void initAttributes(){};

#endif // BUYER\_H

Catalog.cpp

#include "Catalog.h"

Catalog::Catalog(){

initAttributes();

}

Catalog::~Catalog(){}

void Catalog::initAttributes(){}

Category.h

#ifndef CATEGORY\_H

#define CATEGORY\_H

#include <string>

#include <vector>

class Category{

public:

Category();

virtual ~Category();

void returnAllProperty(){}

void addItem(){}

void removeItem(){}

protected:

private:

int idCategory;

std::string name;

void setIdCategory(int new\_var){

idCategory = new\_var;

}

int getIdCategory(){

return idCategory;

}

void setName(std::string new\_var){

name = new\_var;

}

std::string getName(){

return name;

}

void initAttributes();

};

#endif // CATEGORY\_H

Department.h

#ifndef DEPARTMENT\_H

#define DEPARTMENT\_H

#include <string>

#include <vector>

class Department{

public:

Department();

virtual ~Department();

void viewPeopleList(){}

void addTask(){}

protected:

std::string worker;

void setWorker(std::string new\_var) {

worker = new\_var;

}

std::string getWorker(){

return worker;

}

private:

std::string name;

std::string task;

int peopleCount;

void setName(std::string new\_var) {

name = new\_var;

}

std::string getName() {

return name;

}

void setTask(std::string new\_var){

task = new\_var;

}

std::string getTask() {

return task;

}

void setPeopleCount(int new\_var){

peopleCount = new\_var;

}

int getPeopleCount(){

return peopleCount;

}

void initAttributes();

};

#endif // DEPARTMENT\_H

Empluyee.h

#ifndef EMPLUYEE\_H

#define EMPLUYEE\_H

#include "User.h"

#include <string>

#include <vector>

class Empluyee : public User{

public:

Empluyee();

virtual ~Empluyee();

void lookSalesList(){}

protected:

private:

std::string salesList;

void setSalesList(std::string new\_var) {

salesList = new\_var;

}

std::string getSalesList(){

return salesList;

}

void initAttributes();

};

#endif // EMPLUYEE\_H

Firma.cpp

#include "Firma.h"

Firma::Firma(){

initAttributes();

}

Firma::~Firma(){}

void Firma::initAttributes(){}

Firma.h

#ifndef FIRMA\_H

#define FIRMA\_H

#include <string>

#include <vector>

class Firma{

public:

Firma();

virtual ~Firma();

void info() {}

protected:

private:

std::string name;

int yearOfFounder;

int INN;

void setName(std::string new\_var){

name = new\_var;

}

std::string getName(){

return name;

}

void setYearOfFounder(int new\_var){

yearOfFounder = new\_var;

}

int getYearOfFounder(){

return yearOfFounder;

}

void setINN(int new\_var) {

INN = new\_var;

}

int getINN(){

return INN;

}

void initAttributes();

};

#endif // FIRMA\_H

Payment.cpp

#include "Payment.h"

Payment::Payment(){

initAttributes();

}

Payment::~Payment(){}

void Payment::initAttributes(){}

Payment.h

#ifndef PAYMENT\_H

#define PAYMENT\_H

#include <string>

#include <vector>

class Payment{

public:

Payment();

virtual ~Payment();

void pay() {}

protected:

private:

std::string type;

bool checkPay;

void setType(std::string new\_var) {

type = new\_var;

}

std::string getType() {

return type;

}

void setCheckPay(bool new\_var) {

checkPay = new\_var;

}

bool getCheckPay(){

return checkPay;

}

void initAttributes();

};

#endif // PAYMENT\_H

Property.cpp

#include "Property.h"

Property::Property(){

initAttributes();

}

Property::~Property(){}

void Property::initAttributes(){}

Property.h

#ifndef PROPERTY\_H

#define PROPERTY\_H

#include <string>

#include <vector>

class Property{

public:

Property();

virtual ~Property();

void info(){}

void newProperty(){}

protected:

private:

int idProperty;

std::string name;

int cost;

int size;

std::string adress;

void setIdProperty(int new\_var) {

idProperty = new\_var;

}

int getIdProperty(){

return idProperty;

}

void setName(std::string new\_var){

name = new\_var;

}

std::string getName(){

return name;

}

void setCost(int new\_var) {

cost = new\_var;

}

int getCost() {

return cost;

}

void setSize(int new\_var) {

size = new\_var;

}

int getSize() {

return size;

}

void setAdress(std::string new\_var){

adress = new\_var;

}

std::string getAdress(){

return adress;

}

void initAttributes();

};

#endif // PROPERTY\_H

User.cpp

#include "User.h"

User::User(){

initAttributes();

}

User::~User(){}

void User::initAttributes(){}

User.h

#ifndef USER\_H

#define USER\_H

#include <string>

class User{

public:

User();

virtual ~User();

void authorization(){

protected:

private:

int userId;

std::string password;

std::string loign;

std::string email;

void setUserId(int new\_var){

userId = new\_var;

}

int getUserId(){

return userId;

}

void setPassword(std::string new\_var){

password = new\_var;

}

std::string getPassword(){

return password;

}

void setLoign(std::string new\_var){

loign = new\_var;

}

std::string getLoign() {

return loign;

}

void setEmail(std::string new\_var){

email = new\_var;

}

std::string getEmail() {

return email;

}

void initAttributes();

};

#endif // USER\_H